**Proceedings of the**
**1999 ASME Design Engineering Technical Conferences**
**September 12-15, 1999, Las Vegas, Nevada**

# DETC99/DAC-8597

## A LEAN-SERVER APPROACH TO ENABLING COLLABORATION USING ADVANCED DESIGN METHODS

**Dr. Mark A. Hale**[1] **Dr. James I. Craig** **Dr. Dimitri N. Mavris**
Georgia Institute of Technology
School of Aerospace Engineering
Atlanta, GA 30332-0150

**KEYWORDS:** Collaboration, Design, Internet, Server

### ABSTRACT

The design framework research community utilizes the internet as a facilitator for collaborative activities. The internet provides a platform independent interface and geographic distribution. Information can be easily represented to the end-user using the HyperText Markup Language and can be coded in a reasonable timeframe. A shortcoming of exploiting this technology further is the reliance on existing Web servers to interface with analysis tools and design services. In this case, interactive components require auxiliary processes (called CGI scripts) to be started by the server that are used in an inefficient manner. This model does not provide the user-oriented capability required by standalone applications. Java offers improvements in client-side processing but a server bottleneck still exists. A novel approach using lean-servers is introduced as an *alternative* method for providing an efficient server-side computing model. In this case, internet requests are brokered directly by the design application by providing a gateway to the application's programming interface using a HyperText Transfer Protocol compliant layer. This allows requests to be managed directly by the application rather than requiring auxiliary services. Step-by-step directions for implementing this approach using a case study of an existing design framework are given in this paper. A prototype system, called the Systems Programming Architecture for Collaborative Engineering, is described as one scenario for implementing the lean-server technology. Usability of the approach is demonstrated through a Design of Experiments example that is representative of modern design methods. This example also demonstrates collaboration because it executes asynchronously with multi-user intervention at any time during the process. In hindsight, the lean-server approach is an enabling technology for collaborative design and focuses future research direction on the establishment of collaborative design practices.

### NOMENCLATURE

| | |
|---|---|
| API | Application Programming Interface; a mechanism for accessing internal functions within an application |
| CGI | Common Gateway Interface; a mechanism for running auxiliary processes on Web servers |
| Client | A customer for internet services; a web browser |
| Collaboration | The act of working together |
| HTML | HyperText Markup Language; a portable language for describing pages of mixed text and images with embedded references |
| HTTP | HyperText Transfer Protocol; a protocol for information exchange on the Web |
| IMAGE | Intelligent Multidisciplinary Aircraft Generation Environment; a design framework developed at Georgia Tech |
| Server | Distributes information in response to internet requests |
| Socket | An endpoint of communication to which a name may be bound; facilitates inter-process and network communication |
| SPACE | Systems Programming Architecture for Collaborative Engineering; a demonstration framework incorporating IMAGE and lean-server technologies. |

[1] Corresponding Author

Dr. Mark A. Hale
Georgia Institute of Technology
School of Aerospace Engineering
Atlanta, GA 30332-0150
404.894.9810
404.894.6596 Fax
mark.hale@ae.gatech.edu

## INTRODUCTION

The notion of collaborative design conjures a grand vision of many designers, working at geographically distributed locations, making good design decisions in a reduced timeframe, realizing a lower product cost. In fact, this is the vision of NASA's current Intelligent Synthesis Environment initiative.[1] Taking a closer look, collaborative access to engineering analyses is one of the many sub-tasks which plays a vital role in the engineering design process. Turning this into reality leads to the following question:

> *What enabling technologies are needed for collaborative access to advanced design methods?*

More specifically, how can such technologies support the following objectives during collaborative exercises:

- Concurrent access to applications
- Coordination of resources
- Distillation of information from distributed sources
- Deployment of advanced design methods
- Reasonable interface response times
- Transparency of technology to end user
- Minimum maintenance
- Lower cost

There are a number of possible internet technologies that exist to facilitate collaboration and two will be discussed within the context of this paper. The authors go on to introduce an alternative, called the lean-server approach, that is consistent with the expected performance of the technology. This will be highlighted through a Design of Experiments example conducted using a Web-based interface.

The World-Wide-Web provides global internet services for information transactions and is an enabler for collaborative design practices. The successes of the Web can be attributed to affordable, common, platform independent interfaces that make computer transactions transparent. The Web consists of client and server applications operating on the internet as shown in Figure 1. In the client-server model, the number of clients is presumed to be far greater than the number of servers. HyperText Markup Language (HTML) documents are managed and distributed by Web servers using HyperText Transfer Protocol (HTTP). Client applications (browsers) load documents from the respective servers and make them available to the user. As a result, the interface is logically separated from the application.

Servers can also process application requests from the client using the Common Gateway Interface (CGI). This allows the server to make use of server-side processes for handling client requests. For example, information can be returned from the client page (for instance, from a form-based page) and processed by a server-side application operating through the CGI. The server then returns the application response to the user.

Java™ adds client-side processing capabilities to the client-server model. These are called applets and are depicted by the Sun Java™ logo in Figure 1. Applets can be run within Web browsers, as well as independent applications, and provide diverse functionality: including graphing, forms with local calculations, and animation with local rendering. These applets are downloaded from HTTP servers similar to the manner in which HTML documents themselves are downloaded. Java™ applets are stored and transferred in byte form to preserve source code but remain platform independent. JavaScript™, developed by Netscape Corporation, is similar to Java™ in that it adds client side computing. JavaScript™ is contained as text within HTML documents rather than transferred separately in byte form.

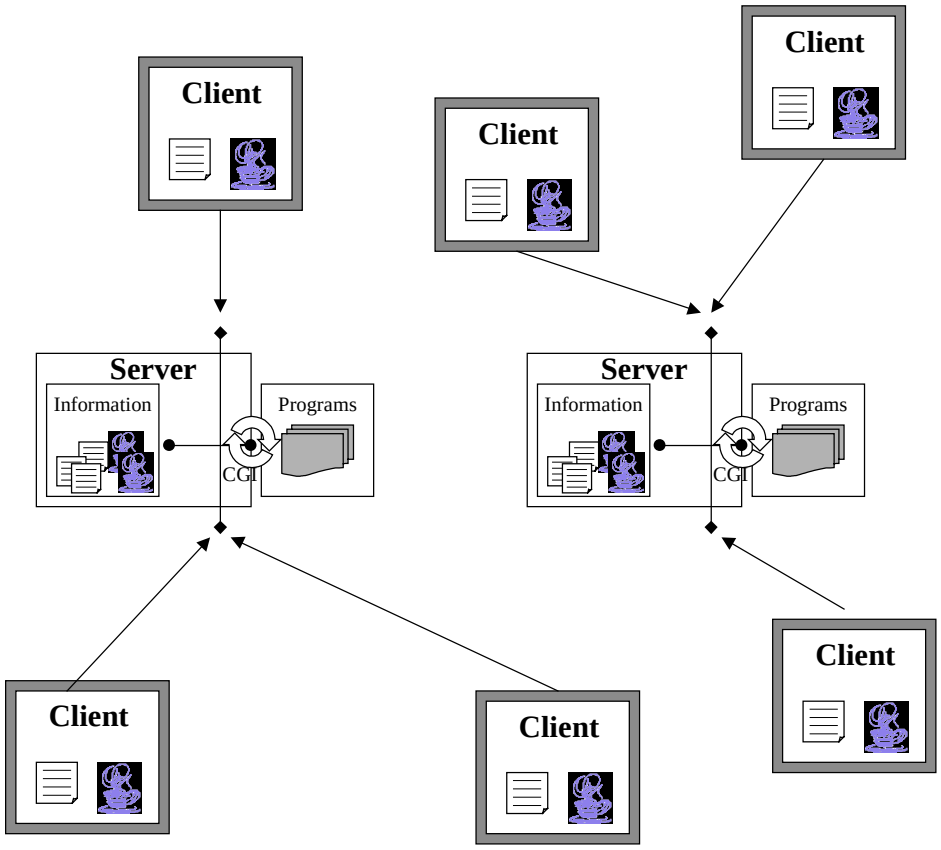



**Figure 1. Internet Client-Server Configuration: The Status-Quo**

## BACKGROUND

This paper focuses on the utilization of systems and analysis applications for collaborative design on a ubiquitous computing network. There are two popular Web methods for enabling distributed applications: CGI and Java™ technology. These will be outlined here as a starting point for implementing collaborative design principles. They represent a composite example of technology implementations and allow for generalizations to be made. It should be noted that there must be more than a distributive component in the final implementation to enable collaborative design.[2] A design management and guidance system is necessary to accomplish design tasks. CGI scripts fail to access such a design system efficiently because these systems are persistent, distributed, and dynamic. A lean-server approach is introduced here as an alternate approach for accessing design software using the client-server paradigm. The CGI and Java™ methods will be described first.

### CGI-Based Design Implementation

Clients can perform server-side processing by posting data to the server that is to be processed by a CGI script. The server initiates execution of the process on the server's file system as shown in Figure 2. This process may communicate with a larger design system or use other mechanisms, such as files, for performing analyses. These CGI processes return properly formatted HTML data for the server to return to the client. The server will not return until the CGI process has completed. The CGI implementation scheme on the server is a common test platform for distributed computing. For example, the Integrated Design System developed at NASA Langley and Ames and the Space Analysis Interface use CGI technology to demonstrate space systems design by geographically distributed end-users. [3,4]

Even though structured, multi-user applications can be developed, the CGI method encounters problems when scaling to the larger vision of a collaborative design system. In a standard server configuration, the CGI processes:

- fail to distinguish separate user process and file spaces,
- respond slower due to the added overhead of starting a new process, and
- are more difficult to maintain because of the custom coding needed to stitch together processes which adds complexity.

Most implementations rely on a single file system for data transfer rather than accessing a design management system, and therefore do not fit a distributed analysis and process model.

### Java™-Based Design Implementation

The use of Java™ applets is another mechanism for providing component distribution. Java™ is more flexible than the CGI because it allows for client-side computing and custom user interfaces. Applets are loaded from the central server in much the same manner in which a standard Web page is loaded. As shown in Figure 3, applets are able to communicate directly with the server rather than relying on client browsers. Applets have been used in some early design environment implementations and have proven to be useful because they introduce different media forms for visualizing design information.[5,6]

The Java™ computing model also presents difficulties as a collaborative design implementation because of performance issues associated with the download of byte code, the maintenance of user-side software, and reliance on auxiliary processes for communicating with design management systems.

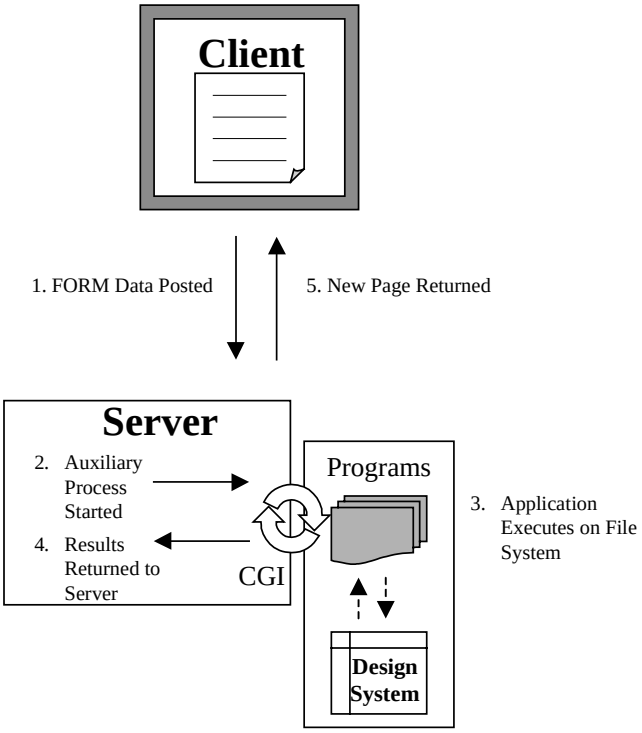



**Figure 2. Details of CGI Process for Processing Form Data**

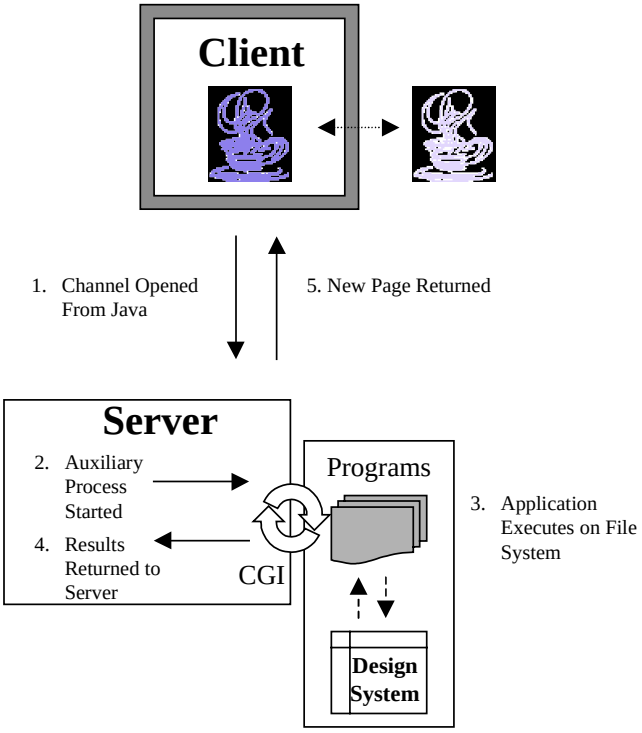



**Figure 3. Details of CGI Process for Processing Java™ Requests**

## THE LEAN-SERVER APPROACH

The lean-server approach is an embodiment of the server directly within a persistent design framework as illustrated in Figure 4. It is designed to maintain the performance of client-side browsing and the tractability of Java™. The client connects to a server process that is imbedded directly in the design application so that no auxiliary processes are required. This approach allows one to overcome a number of the shortcomings in other implementations when accessing the application resources during design. The lean-server approach concept originates from the need to provide a level of design management overlooked in early Web demonstrations of engineering analysis and design frameworks. This paper focuses on the technology's application for implementing design frameworks as servers but can be used for individual analyses as well.

### Direct Access to a Design System

Design systems and analysis tools behave as servers and respond to requests that come across the internet. In this manner, the design system can be left running as though it were being used as a stand-alone, single-user application. This capability permits data and state to be maintained within the system. The connectivity to the Web can be kept simple because the additional overhead of a dedicated Web server and custom CGI scripts is not needed. Furthermore, the same *validated* design system is used to respond to internet requests as in the single-user system. The authors have found that some interface features (such as drag & drop) cannot be achieved in the standard Web clients and are more easily done in the standalone applications.

### Collaborative Capability

The lean-server approach facilitates the distributed component of collaboration. Web requests may come from any internet enabled Web client. These ubiquitous requests may come from simultaneous users interacting to solve a design problem. In addition, autonomous analyses and agents may also be involved in problem solving. The design problems may take years to solve at the macro level and hours to days for technology investigation. At any time during this process, players can come into and out of the process. In order to achieve this level of collaboration, a shift from single-user, synchronous evaluation to multi-user asynchronous problem solving must occur. The Design of Experiments example described at the end of this paper illustrates how this can be achieved.

### Bridge from Client-Server and Peer-Peer Models

The lean-server approach also bridges client-server and peer-peer simulation strategies. The client-server method is made popular by the abundance of internet browsers running on personal computers and are used readily for on-line searches, commerce, and travel reservations. Peer-peer computing models are used in simulation federations such as those using Department of Defense's High Level Architecture (HLA).[7] This model emphasizes the simulation that must occur on each node and the inter-node communications and coordination. A simulation node in the peer-based architecture can behave as a server in client-server architectures using the lean-server approach. This multi-role capability is illustrated in Figure 4. This role is being investigated as a method for scaling the Web demonstrations discussed in the background to full-scale enterprise simulations.

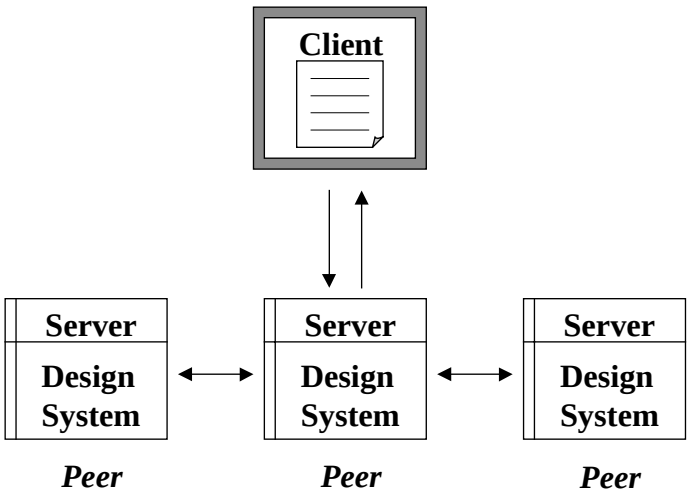



**Figure 4. The Lean-Server Approach**

The lean-server approach brings the technologies needed for collaboration to the research forefront.

## LEAN-SERVER IMPLEMENTATION

The process by which a lean server can be added to design systems will now be described. Specific implementation technologies have been selected for illustrative purposes. Other technologies can be substituted to obtain the desired server configuration or a new system can be structured around the interfaces presented here.

### Step 1: Base Application

The premise of the lean-server technology is that design-oriented applications and systems (and their associated knowledge) are persistent – their services are needed for more than the duration of a single Web transaction. There are a number of stand-alone design frameworks available on the market that are suitable a baseline technology for a web-based framework. The approach described here can be applied to these legacy applications, new applications, and individual analysis tools.

The steps in implementing the lean-server approach will be highlighted using a portion of the designer-centered framework developed at Georgia Tech called the Intelligent Multidisciplinary Aircraft Generation Environment (IMAGE).[8] IMAGE is typical of design frameworks being developed in its class and includes a data modeler, process modeler, analysis interface, and built-in design tools. The architecture is used for research studies in the area of complex systems design in Georgia Tech's graduate design program.

IMAGE is implemented with the Toolkit for Tool Command Language (Tk/tcl) programming suite and its extensions.[9,10] The software architecture is comprised of components organized into several levels of abstraction accessible through the graphical tools or through an external programming interface. Specific programming techniques and packages may vary with other framework implementations; however, the components are analogous and comparisons are valid in the context of this paper. In addition, the lean-server technology is equally as applicable to stand-alone analysis applications.

## Step 2: Application Programming Interface

Mature software applications provide interfaces for end users to access internal routines.[11] An Application Programming Interface (API) permits this and is often used to customize an application. This type of interface provides a straightforward mechanism for implementing the lean server concept. Some API examples include writing an import utility to read a corporate proprietary dataset, adding an optimizer to the base library of advanced design methods, and scripting common tasks. Example API's include, Macros/VisualBasic in Microsoft Excel™, CATGEO in CATIA™, and M-Files in MATLAB™. The API for IMAGE is designed to provide access to the underlying software architecture so that the user can provide custom interaction with the database, process management, design capability, and other functions. To illustrate the API, the code required to implement a geometrical object used in the object-oriented system in IMAGE is shown in Code Segment 1. The class is named `FORM` and is written using the tcl language and requires an object-oriented extension to tcl called itcl to implement.[12]

An API permits the user to use these base classes, procedures, and functions for building a customized interface. An example of the use of an API to create the aircraft object hierarchy shown in Figure 5 using the `FORM` class is shown in Code Segment 2. Objects are assigned a location in the hierarchy as they are created with the `FORM` command. As the objects are created, each object is given a unique Object IDentification (OID) number as returned with the `NewChild` command.

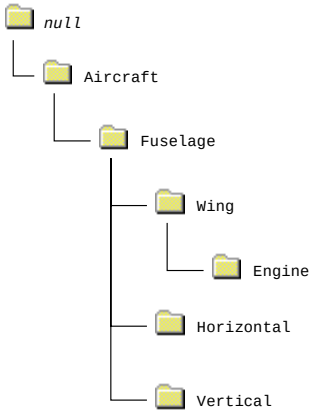



**Figure 5. Aircraft Hierarchy**

## Step 3: Adding HTTP Server Capabilities

Server capabilities must be integrated into IMAGE so that client requests can access the API. This translates into the addition of a socket capable of brokering HTTP requests. HTTP/1.0 URL-encoded requests are structured into the following form by the web browser:

```
GET /command HTTP/1.0
```

This command is sent from the client through a socket to the server. To add these services to IMAGE is relatively straightforward because a communications layer with API already exists. The commands to add an HTTP layer to IMAGE are shown in Code Segment 3. First, a procedure called `WebSOCKET` is created to process the request. Second, a socket is created and bound to the procedure. IMAGE can now be accessed from a web browser using the following URL:

```
http://www.asdl.gatech.edu:5000
```

```
#
#  FORM class
#  Description:  This class is a container for an
#                objected-oriented model where
#                each form instance is assigned
#                a parent and child
#
class FORM {
    constructor (args) {
        # Assign arguments
        eval configure $args
        # Create a new schema instance and assign
        # it to this object
        set Schema [Schema [OID] –Form $this]
        $parent AddChild $this
        return
    }
    destructor {}
    method AddChild (oid) {
        # Add the Child to the list
        lappend $Children $oid
        return $oid
    }
    method NewChild (name) {
        # Create a new FORM object
        set Child [FORM [OID] –Name $name \
            -Parent $this]
        # The new object is a child
        $this AddChild $Child
        # Return the new Object ID
        return $Child
    }
    ...
    public variable Name “unknown”
    public variable Parent null
    public variable Children null
    public variable Schema null
}
```

**Code Segment 1. FORM Class in Object-Oriented System**

```
set aircraft [OID]
FORM $aircraft –Name "Aircraft" –Parent null
    set fuselage [$aircraft NewChild “Fuselage”]
        set wing [$fuselage NewChild “Wing”]
            set engine [$wing NewChild “Engine”]
        set htail [$fuselage NewChild “Horizontal”]
        set vtail [$fuselage NewChild “Vertical”]
```

**Code Segment 2. Use of API to Create Objects**

Within the `WebSOCKET` procedure, the client requests (or commands) will be processed with the `WebProcessRequest` procedure. As each request is processed, a new web page is returned to update the client browser.

## Step 4: Serving HTML Through the API

Direct access to API functions through internet channels is the basis for the lean-server concept. The addition of HTTP services to IMAGE as outlined earlier provides a method for establishing protocol. The `WebProcessRequest` procedure is used to access the API. For example, assume that a web page exists that depicts the aircraft geometry structure as shown in Figure 5. The web page is shown in Figure 6 and is accessed through the following URL:

```
http://www.asdl.gatech.edu:5000/
display FORM
```

The URL references the lean-server enabled IMAGE. The command "`display FORM`" is an API to return a tree view of the FORM hierarchy in HTML.

The plus sign immediately below each object has a hyperlink associated with it and is used to create a new child object. For instance, the hyperlink corresponding to the plus sign below the Wing object has the hyperlink reference:

```
<A HREF="http://www.asdl.gatech.edu:5000/
880635-2217 NewChild Sub-System;display
FORM">+</A>
```

The reference to the IMAGE lean-server is given in the `HREF` as:

```
http://www.asdl.gatech.edu:5000
```

The command to create a new object is:

```
880635-2217 NewChild Sub-System
```

The Wing object has a unique identification number of `880635-2217` created from a call to the `OID` procedure during the generation of the page. The final command:

```
display FORM
```

is an API command that will update the browser with the additional object in the hierarchy immediately below the `Wing`.

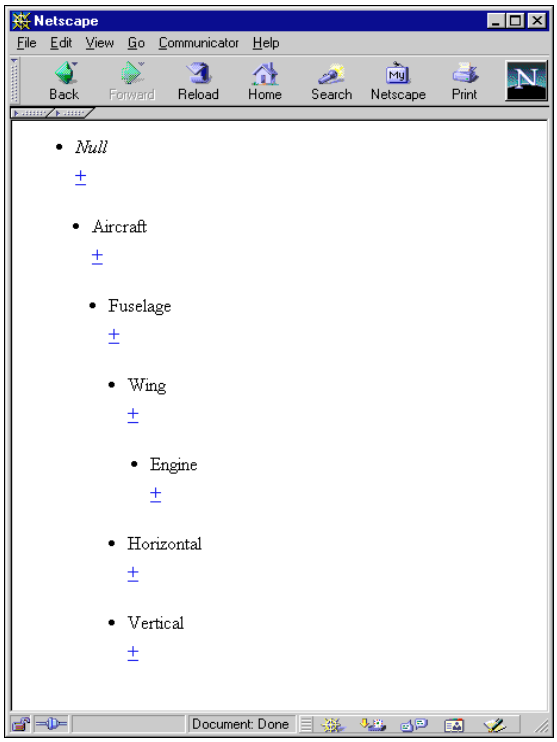



**Figure 6. Aircraft Form Hierarchy in Web Page**

```
#
# Create procedure to be called
# when an internet request comes in
#
proc WebSOCKET {sock args} {
    #
    # Get requests
    #
    set GET [gets $sock]
    #
    # Process incoming requests in the form
    # GET /command HTTP/1.0
    #
    # Make sure using HTTP/1.0
    if { [string range $GET \
        [expr [clength $GET]-8] end] \
        != "HTTP/1.0" } {
            flush $sock
            close $sock
            return
    }
    # Pull out the request
    set request [string range $GET 5 \
        [expr [clength $GET]-10]]
    #
    # Process incoming requests
    # Results will be in html format
    # Return the header first
    #
    puts $sock "HTTP/1.0 202"
    puts $sock "Content-type: text/html"
    puts $sock ""
    WebProcessRequest $request
    #
    # Finish processing
    #
    flush $sock
    close $sock
}
#
# Create a socket and assign a port number
#
socket "myaddr www.asdl.gatech.edu" server \
    WebSOCKET 5000
```

**Code Segment 3. Addition of HTTP Services**

## EXTENDING THE APPLICATION API

An advantage of using the client-server approach is that it separates the user interface from the application. This separation provides a mechanism for compatibility and maintainability as both technologies mature and evolve. Current Web browsers can transact with lean-server enabled applications using a variant of HTML over HTTP. The API example described earlier demonstrated the invocation of API commands directly by client applications. If the API provides a response to the user, a new client page is displayed.

The API used in the IMAGE environment, which provides the baseline example for this paper, was extended to include additional commands for producing HTML documents. The API includes the HTML commands shown in Table 1. The API was configured so that the interface designer can utilize HTML macros written in tcl rather than have to write pure HTML. Consider the example shown in Figure 7. The macro syntax is more compact and is translated into HTML as the request is served. This enables the interface designer to mix programming

constructs with the HTML macros allowing for context sensitive HTML responses. Furthermore, the macros can be configured to return their responses based on the client application. The discussions thus far have presumed that the client is a Web browser. If the client was instead Microsoft Excel™ as shown in the figure, the macros can easily be expanded to return a file with Comma Separated Values (CSV) so that the table can be reproduced in Excel.

**Table 1. HTML Tags used in Lean-Server Implementation of IMAGE**

| | |
|---|---|
| TITLE | FONT |
| BODY | FRAME, FRAMESET |
| H1,H2,H3 | TABLE, TR, TD, TH |
| I, B | IMG |
| A | UL, LI |
| HR | DL, DT, DD |
| CENTER | FORM, OPTION, INPUT, |
| HEAD | SELECT, TEXTAREA, |
| HTML | LABEL |

A shortcoming of the Netscape™ and Internet Explorer™ browsers tested with the lean-server demonstration is that the server is required to deliver a return page to display in the browsers. If an API is invoked that has no return information, for example when updating the value of a variable in a form field, the browsers return an error that the server had unexpectedly closed the connection.

## BENCHMARK

A small application has been constructed using a subset of the IMAGE Version 1.17 API, HTML macros for the API, and lean-server implementation. This application has been termed SPACE (Systems Programming Architecture for Collaborative Engineering). The application was benchmarked against the popular Apache Web Server Version 1.2.4 to validate the baseline server performance. The user's home page for SPACE as shown in Figure 8 and a sample form-based page as shown in Figure 9 were tested with each server. The home page contains two frames, ad 5 distinct GIF files. The form-based page contains 10 input elements. Care was taken to reproduce the pages on the Apache Server and was needed to be done because the SPACE application serves its pages dynamically – the lean-server approach. Both SPACE and the Apache Server were run on a DEC Alpha 2000 5/300 workstation running OSF V4.0.

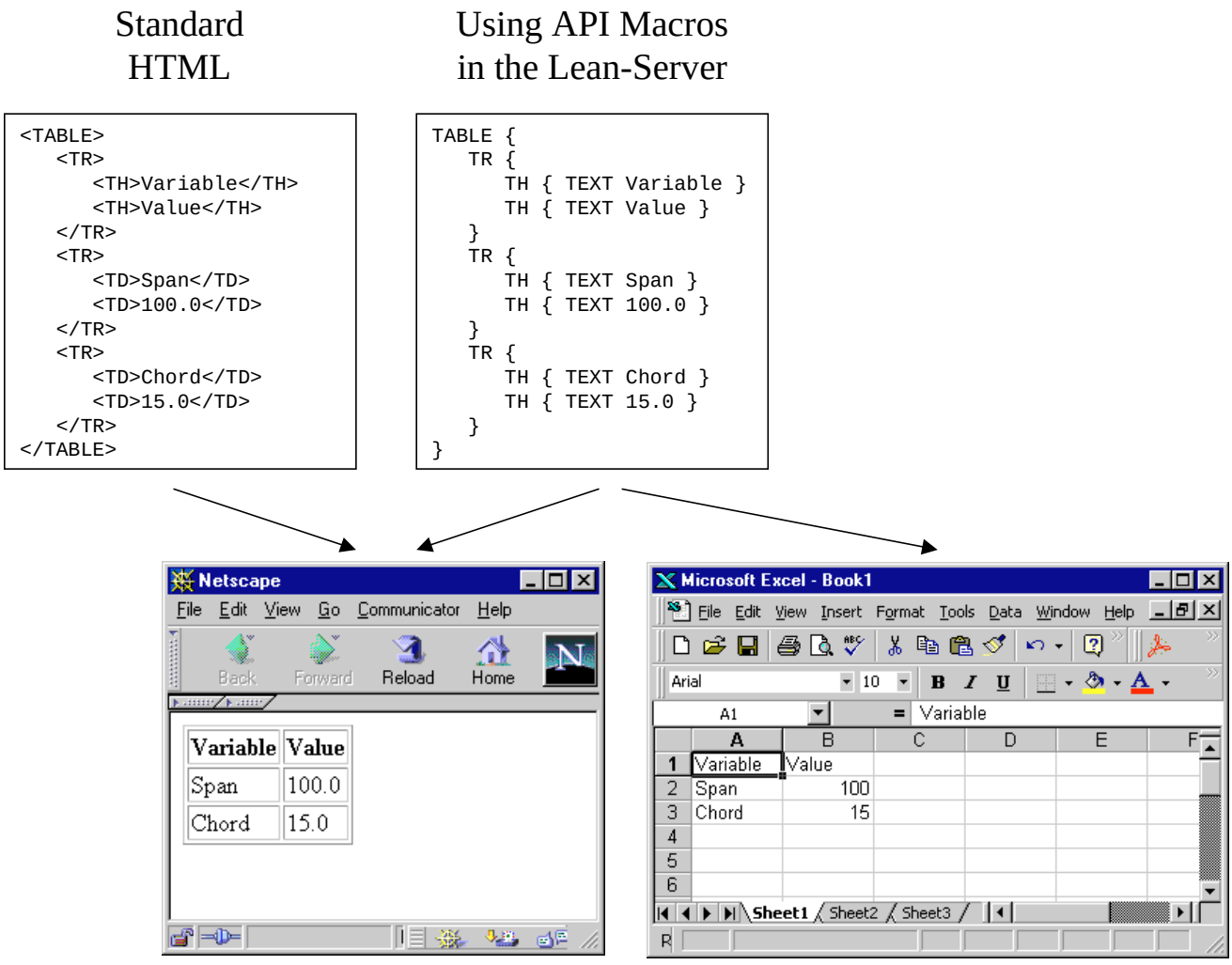



**Figure 7. API for HTML Macros**

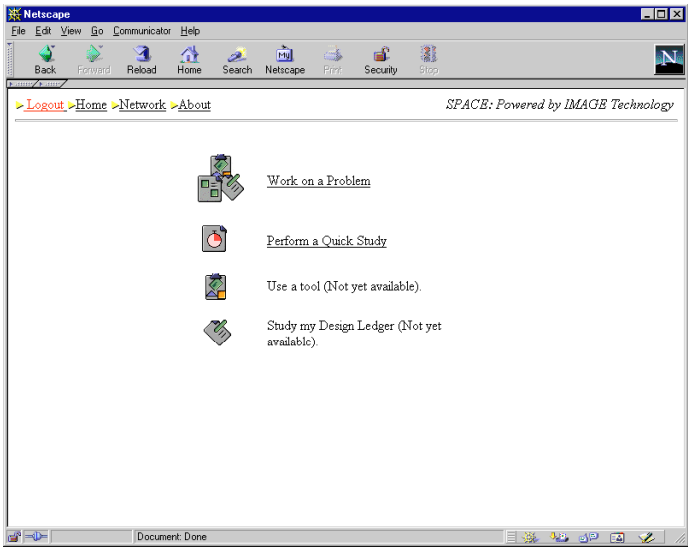



**Figure 8. SPACE User's Home Page**

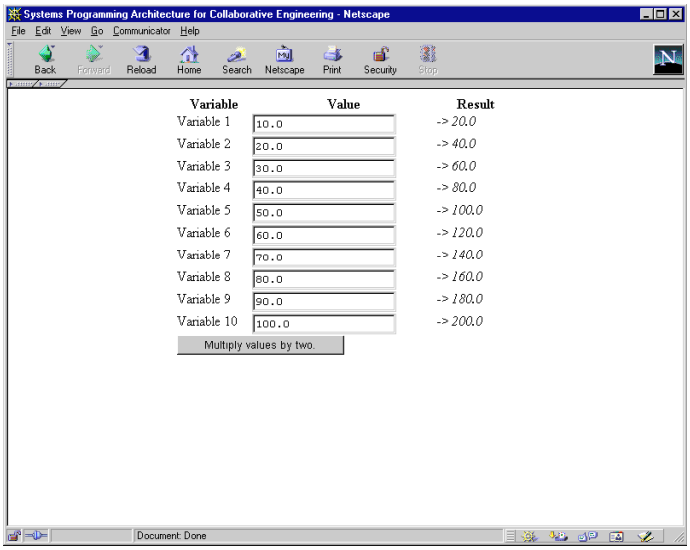



**Figure 9. FORM-Based Web Page**

The Ziff-Davis WebBench™ 2.0 software was used to perform the benchmark tests. The benchmark clients were executed on a 500MHz Pentium III PC running Windows NT™ for a duration of one minute (connection times were found to be on the order of hundredths of a second). Both single client and multiple client connectivity were tested. The results of the benchmark are shown in Figure 10 and Figure 11. The lean-server provided similar performance in all but the delivery of HTML pages to multiple clients. This result was expected because the lean-server implementation was not optimized for multiple connections whereas the Apache server has been modified for Web performance. These preliminary benchmark results indicate that the use of the lean-server approach does not lead to a degradation in internet performance. With this in mind, an example highlighting the desired functionality of the lean-server approach will be discussed next.

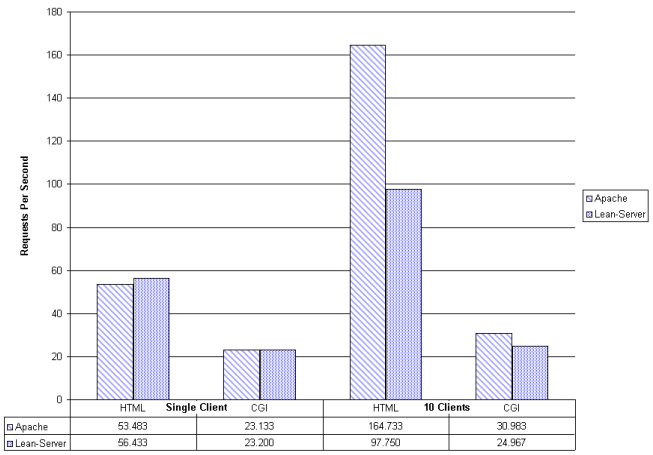



**Figure 10. Benchmark: Requests Per Second**

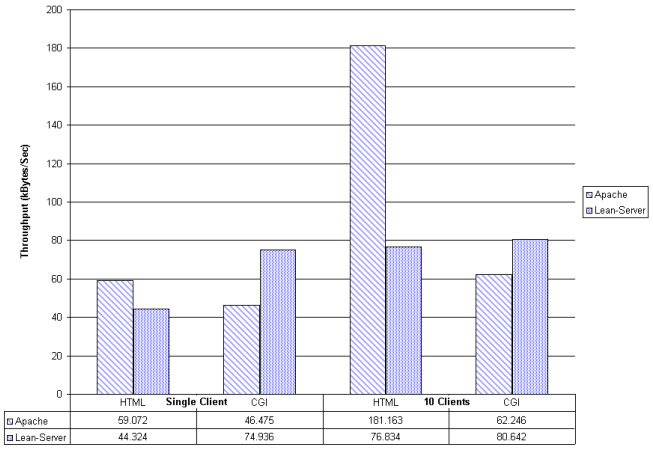


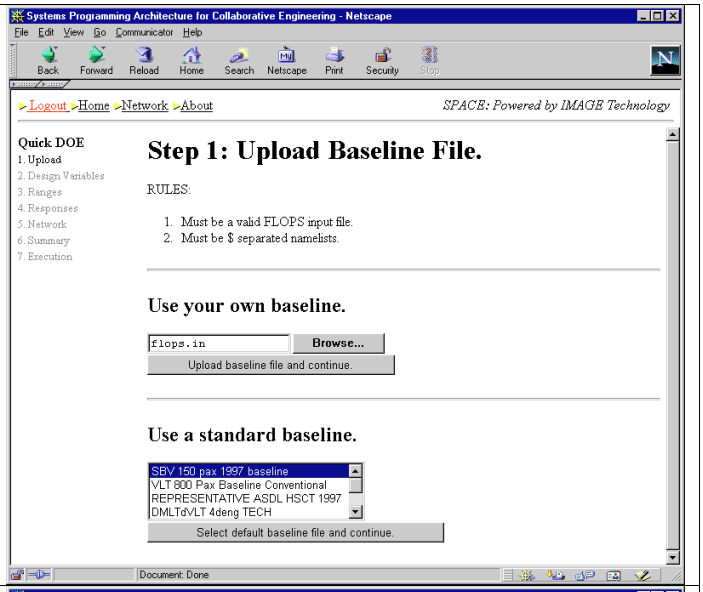

**Figure 11. Benchmark: Throughput**

## APPLICATION TO DESIGN OF EXPERIMENTS

The SPACE application used in the benchmark is also used to highlight the functionality of the proposed lean-server approach. A scenario involving a Design of Experiments is illustrated here. The Design of Experiments can be used to form design-oriented approximations that represent sophisticated analyses in compact form.[13] This design example is representative of modern design tools making their way into standard design practice. In the example, a designer has connected to SPACE via a Netscape™ browser, logged in, and selected to due a "Quick Design of Experiments" which involves a pre-defined performance analysis using FLOPS, a synthesis and sizing tool developed at NASA Langley.[14] The series of prompts that are displayed to the user, via the lean-server, are shown in the following screen capture sequence.

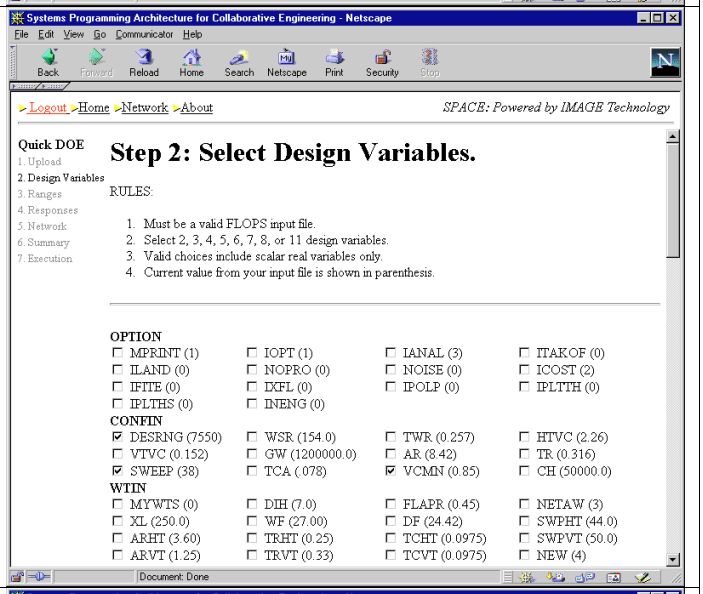
## DOE Step 1

The user is directed to upload a custom baseline file (FORTRAN Namelist) for which a Design of Experiments will be performed or the user can select from a series of default vehicle configurations. The user is given the option to make modifications to the input file if a default configuration is selected. In this example, a rapid performance evaluation of a Very Large Transport aircraft is done using the sizing and synthesis tool FLOPS. The baseline file is uploaded in the "Use your own baseline" field.



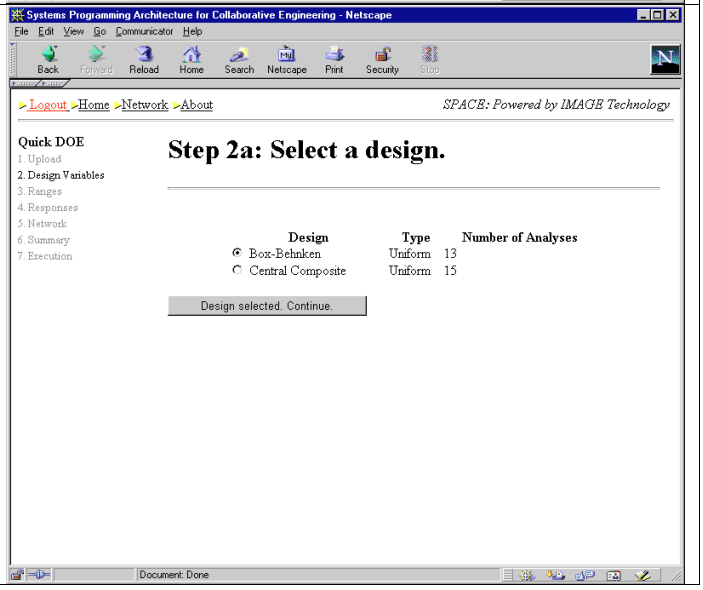

## DOE Step 2

Design variables are extracted from the Namelist input file and displayed. The user can select 2,3,4,5,6,7,8, or 11 variables for which Design of Experiment tables exist in the system (others are possible but have not been implemented). For this example, the design range (DESRNG), wing sweep angle (SWEEP), and cruise mach number (VCMN) are identified in the configuration FORTRAN namelist (CONFIN).



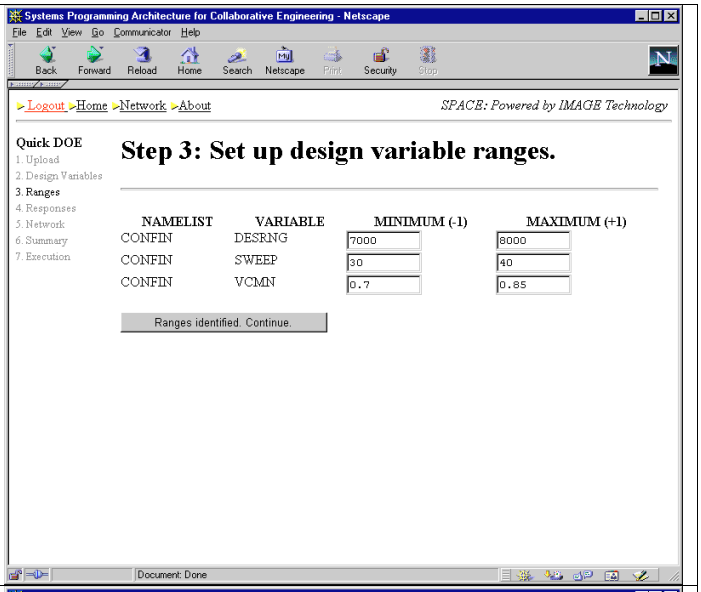

## DOE Step 2a

There may be several alternative experimental designs that can be exercised depending on the number of design variables selected. On this particular screen, the user can choose between a Box-Behnken or Central Composite design. Notice that the Box-Behnken design requires fewer analysis cases.



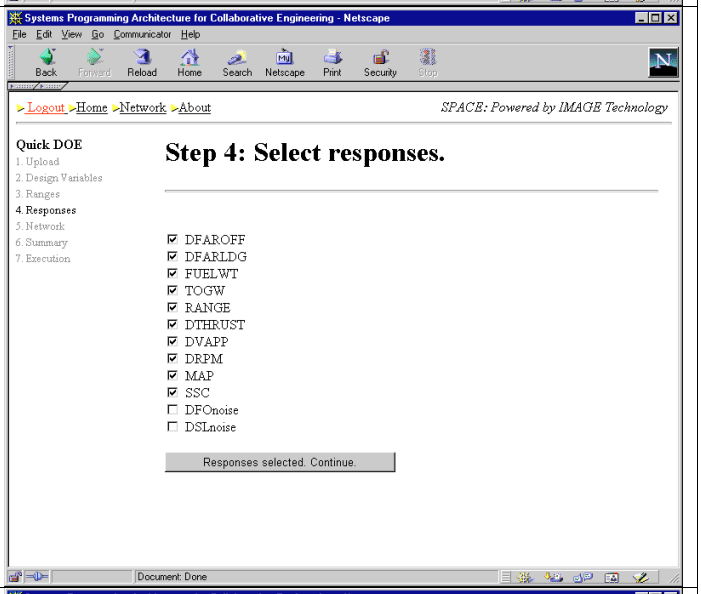
## DOE Step 3

The ranges for which the design variables will be varied in the Design of Experiments are entered next. The values for these variables will be substituted into the baseline file at each case during the execution of the experiment.




## DOE Step 4

The user can track any of the vehicle performance metrics shown to the right. These are the responses that are generated in the Design of Experiments. In this particular example, no aircraft noise calculations are desired. Consequently, the sideline and fly over noise parameters are not recorded.

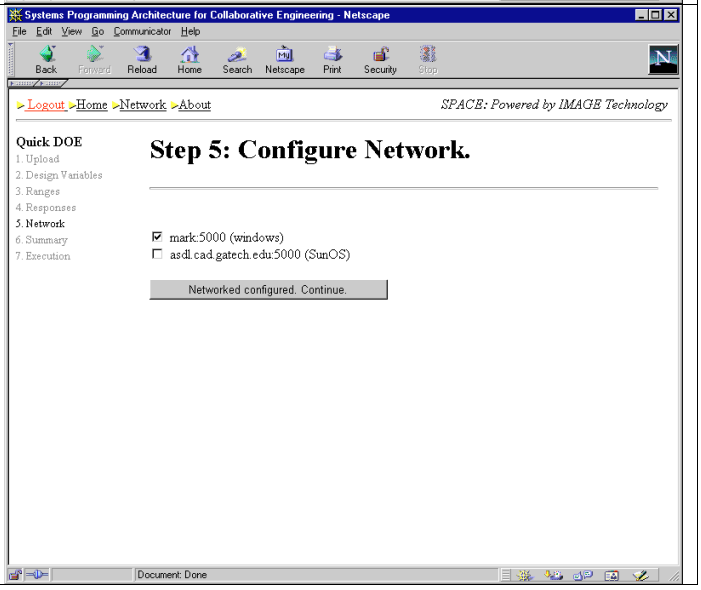



## DOE Step 5

The user selects particular machines from the available network to execute the problem. In this case, the server running under Windows on the same machine that this paper is being written on is selected. The internet address is:

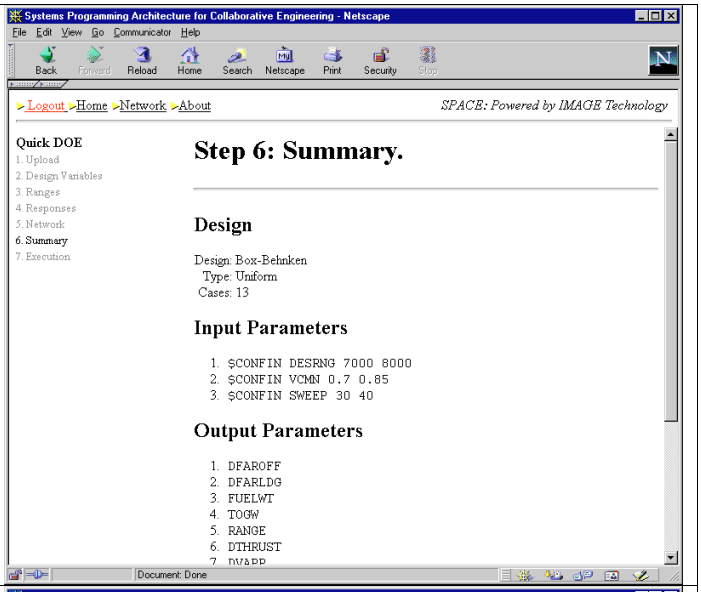
http://mark.cad.gatech.edu:5000

If multiple machines are selected, individual designs will execute in parallel.



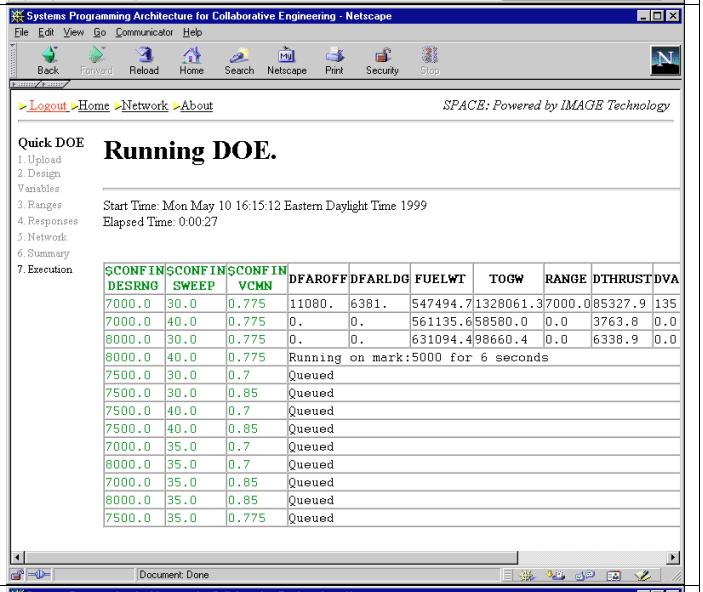
**DOE STEP 6**

The user is presented with a summary before the cases are executed. Although the Web interface is straightforward to use, it is still important to verify the setup before expending computational resources since some experiments may take days to complete. When confirmed, the user selects an execute button just off the bottom of the screen area.



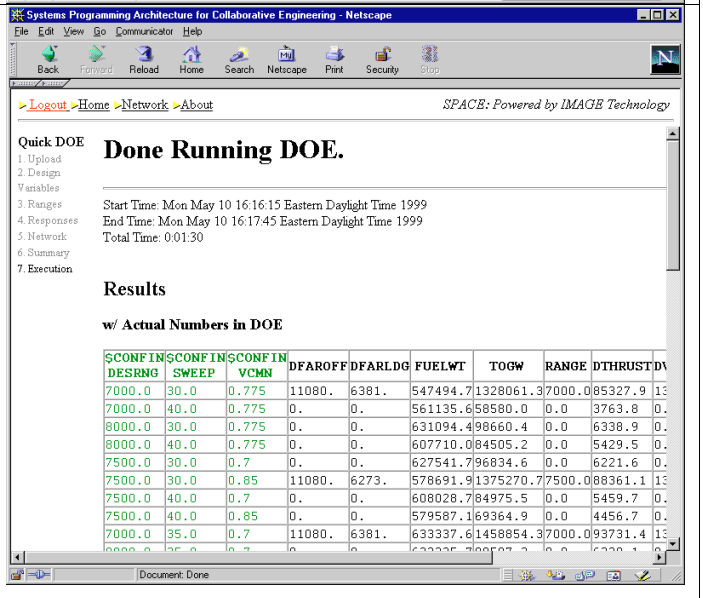

**DOE EXECUTION**

A table showing the current execution status is presented to the user as the experiments are performed. As and experiment is completed, its results are tabulated and another experiment is assigned to the free machine. In this example, only one machine is being used.




**DOE RESULTS**

The user is presented with the experimental results at the completing all of the cases. Execution date and time is given for identification. This data can be copied and pasted directly into post-processing applications. The authors are currently implementing regression analysis to provide immediate feedback on the experiment to the user. Notice that this particular experiment failed to produce valid results, easily seen by the zero range and takeoff and landing field lengths. This interface allowed for a quick investigation into the valid ranges for the experiment which promotes discovery before resource allocation.



This example shows the straightforward application of the lean-server approach. However, it is difficult to distinguish the portion of the interface that is facilitated by the lean-server and that which can be done using CGI scripts. A final screen capture is shown in Figure 12 that distinguishes the lean-server. At any time during the execution of the experiment, the user can log out of the web browser and the problem will continue to execute asynchronously – the problem will run regardless of whether or not the user is managing the problem. To connect to the running problem, a summary is presented as the user logs in to the design server. An already completed and currently running experiment are presented to the user in the example shown in Figure 12. Perhaps not so obvious, any user (not just the one that created the problem) with the correct permissions can connect to these problems from anywhere on the internet and they may do so at the same time. This is just one mechanism for facilitating collaboration. The authors have begun further research into the design method algorithms to operate asynchronously and facilitate collaboration.

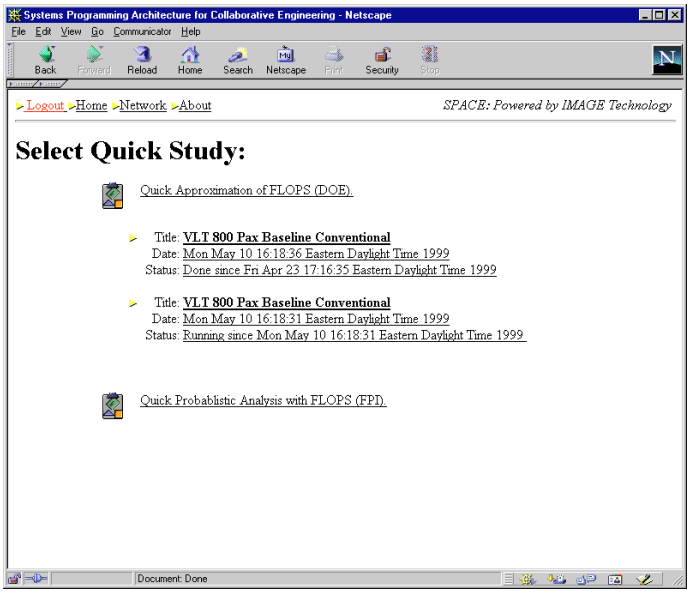



**Figure 12. Tracking Multiple Experiments**

## CONCLUSIONS AND RECOMMENDATIONS

The lean-server approach provides a mechanism for providing internet-based services and design system access for collaborative design. This approach has the following benefits:

- Legacy applications can be extended to service internet transactions with minimal coding requirements
- No auxiliary processes must be used for internet capabilities allowing the servers to be used efficiently and transparently
- Design services can be distributed across a network
- Off –the-shelf technologies are employed and are compatible with legacy client browsers

These advantages were briefly highlighted through an example client interface to a Design of Experiments tool.

The lean-server approach facilitates the distribution of design methods using the internet. Knowing that distribution can be achieved with relative ease, research attention can be given other items important to design collaboration, including multi-player decision-making, management, scheduling, and new methods.

## ACKNOWLEDGMENTS

The authors wish to acknowledge Mr. Dennis Carter of the Air Force Research Laboratory, Mr. Samuel Dollyhigh of the NASA Langley Research Center, Mr. Thomas Galloway of the NASA Ames Research Center, and Ms. Katherine Drew of the Office of Naval Research for research support for this work. The authors also with to thank the developers of the Tk/tcl scripting language for providing a mechanism for the rapid prototyping of new technologies.

The authors also wish to thank the staff and students of the Aerospace Systems Design Laboratory for providing real design feedback for this research.

## REFERENCES

1 Goldin, D. S., S. L. Venneri, and A. K. Noor, “Beyond Incremental Change,” IEEE Computer, pp. 31-39, October 1998.

2 Port, D. and G. Kaiser, “Collaborative Technologies for Evolving Software Systems,” IEEE Computer, pp. 79-83, October 1998.

3 Allen, G.A. Jr., P.J. Gage, and E. Venkatapathy, “A Web-based Analysis System for Planetary Entry Vehicle Design,” 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,. St. Louis MO, September 2-4, 1998. AIAA 98-4826.

4 Acton, D.E. and J.R. Olds, “Computational Frameworks for Collaborative Multidisciplinary Design of Complex Systems,” 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,. St. Louis MO, September 2-4, 1998. AIAA 98-4942.

5 Becker, J. C., Bloebaum, C. L., "Distributed Computing for Multidisciplinary Design Optimization using Java as a Web Interface", 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, September, 1996.

6 Hulme, K. F., Bloebaum, C. L., "Development of CASCADE: A Multidisciplinary Design Test Simulator", presented at the 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, September, 1996.

7 Proposed for Adoption Fall 1999: HLA Rules IEEE 1516.1, I/F Spec IEEE1516.2, OMT IEEE 1516.3

8 Hale, M. A., "An Open Computing Infrastructure that Facilitates Integrated Product and Process Development from a Decision-Based Perspective," Doctoral Dissertation, Georgia Institute of Technology, School of Aerospace Engineering, July, 1996.

9 Ousterholt, J. K., An Introduction to Tcl and Tk. Reading, MA: Addison-Wesley Publishing Company, Inc. 1993.

10 Welch, B. B., Practical Programming in Tcl and Tk, Upper Saddle River, NJ: Prentice Hall PTR. 1995.

11 Hale, M. A. and J. I. Craig, "Techniques for Integrating Computer Programs into Design Architectures," Sixth AIAA / NASA / USAF / ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, September 4-6, 1996. AIAA 96-4166.

12 McLennan, M. J., "Incr Tcl - Object-Oriented Extensions for Tcl," AT&T Bell Laboratories, March 25, 1994.

13 Box, G.E.P., Draper, N.R., Empirical Models Building and Response Surfaces, John Wiley & Sons, Inc., 1987

14 McCullers, L. A., “Flight Optimization System: User’s Guide,” NASA Langley Research Center 1993.